REMARKS/ARGUMENTS

By this paper, Applicant responds to the Office Action of January 13, 2005 and respectfully requests reconsideration of the application, under the terms of the "Request for 'Personal Check' and Request for Appeal Conference" submitted herewith. The shortened statutory period runs through July 13, 2005. Accordingly, this response is timely.

I. Real Party in Interest

The real party in interest is ATI International SRL of Barbados, the assignee of this application. ATI International is related to ATI Technologies, Inc. of Ontario, Canada.

II. Related Appeals and Interferences

Applicant is unaware of any related appeals or interferences.

III. Status of Claims

Claims 1-83 are now pending, a total of 83 claims. Claims 1, 5, 33, 56 and 79 are independent. The only claim that is rejected is claim 5, under § 101. The remaining claims are not allowed, but not rejected either – the examiner has acted outside the scope of the discretion granted him by the Director in withholding allowance of these claims. Because only the Director has the authority to reject claims (35 U.S.C. § 131), and delegates only have such authority as is delegated under the terms he permits (*e.g.*, Forward to MPEP), no other claims are rejected.

A complete copy of the claims is attached hereto as an Appendix.

IV. Status of Amendments

A Rule 116 Request for Reconsideration of March 21, 2005 demonstrated that the claims as currently pending are not rejected and are patentable. In the alternative, the March 2005 paper offered suggested claim amendments to address non-statutory concerns of the Examiner's personal preferences. Because the Advisory Actions of May 4, 2005 and June 1, 2005 mischaracterized the procedural posture of the March 2005 paper, neither the arguments for patentability nor the grounds for entry of the amendment of March 2005 were considered. The Examiner has not "answered all material traversed." Thus, the condition of the application is now unclear.

The Examiner has conceded facts that establish that final rejection is premature.

In the Advisory Action of June 1, 2005, the Examiner concedes that "adding col. 33, lines 1-9 of Chernoff '028 supports and further expands on prior art." This action constitutes a "new ground of rejection" under the definition of that term in *In re Wiechert*, 370 F.2d, 927, 933, 152 USPQ 247, 251-52 (CCPA 1967) ("An applicant's attention and response are naturally focused on that portion of the reference which is specifically pointed out by the examiner. ... [W]hen a rejection is factually based on an entirely different portion of an existing reference the appellant should be afforded an opportunity to make a showing of unobviousness vis-à-vis such portion of the reference," emphasis added).

Neither the January 2005 Office Action nor either Advisory Action states any set of facts that gives the Examiner the authority to close prosecution. Finality is not authorized. Applicant again suggests that the amendments proposed with the paper of March 2005 may be entered, either as of right, or by Examiner's Amendment in the event that the Examiner wishes to satisfy his non-statutory personal preferences.

Thus, Applicant believes that the status of the March 2005 amendments is (a) they are entitled to entry as of right, or (b) in the alternative

On a separate issue, in a telephone interview of July 7, 2005, Supervisory Examiner An stated that **an interview would be appropriate**. Applicant agrees. If any "rejection" remains, **Applicant again requests a telephone interview** with a member of the appeal conference after this paper has been fully considered. Applicant suggests that the best time for such an interview would be immediately after the appeal conference, while the issues are fresh in everyone's mind, before the PTO issues another paper. Applicant also suggests that, since requests for interviews were incorrectly denied, prosecution has not concluded, and should be reopened so that an interview may be conducted.

V. Informal Summary of the Subject Matter

This § V presents a primer in the technology. The subject matter of specific claims is discussed in § VII.

The specification of this application teaches mechanisms for allowing an operating system, such as Microsoft Windows™, to be used with hardware for which that operating system was not specifically designed. For example, the specification teaches how to use Microsoft Windows – which is available only on Intel chips and a very few other processors – on a foreign RISC (Reduced Instruction Set Computer) processor. This allows a totally new hardware design to use a mature and widely-available operating system, even if the source code is not available to allow Windows to be tailored to or customized to that foreign RISC.

A. Background: Multi-Tasking and Context Switches

Since the early 1960's, computers have used "multi-tasking" to rapidly switch between two or more program processes, so that it appears that the computer is running several different programs simultaneously. For example, the computer on your desk allows you to run email, a word processor, and an internet browser simultaneously, and to switch between them rapidly. A user can switch from one program to another, while the first program continues to run.

A switch between programs, called a "context switch," is fairly analogous to switching between two paper-intensive tasks at your desk. When you switch tasks, you must carefully file away all of the papers of the old task and then retrieve the papers of the new task from their files, and lay them out on the desk for efficient access. You must make some record of what you were doing when you set the old task aside, so you know where to resume. Similarly, when a computer switches programs, the full status of the old program must be carefully stored: this typically includes the values of the registers, any descriptors that describe how the processor accesses the memory of the program, and a pointer to the point at which execution is to resume. Then, all of the corresponding information describing the new task (which was previously saved, when that task was switched out) must be loaded, so that control may be transferred to the new task.

Multi-tasking is typically implemented in a computer's operating system, typically at the lowest and most hardware-dependent layers of the operating system. For example, Exhibit 1 (filed March 21, 2005) shows a very precise map for how an IBM mainframe computer stores every relevant piece of state in memory, and reloads new state from memory. Exhibit 2 (also filed March 21, 2005) shows the same kind of information for Intel's Pentium chip as of 1996.

Digital's Alpha has a third form. The three are necessarily entirely different, because the three have entirely different hardware resources.

Therefore, traditionally, an operating system is tied to one precise type of hardware. For example, even within the Intel chip family, operating systems are not interchangeable. One must match the operating system quite precisely to the Intel chip is being used: there was/is a separate version of MS-DOS and of Windows for each new "major" generation of Intel chip. A mismatch will result in a non-functioning system.

B. Background: the Chernoff '028 patent

Chernoff '028 provides a useful concrete example in which to introduce a number of relevant computer design concepts. Generally, Chernoff '028 is directed to allowing programs coded for a non-native computer family, for example, the Intel X86 family, to run on a computer of another family, for example, Digital Equipment Corp.'s Alpha RISC processor (col. 7, line 29). Alpha programs run on Alpha in the conventional way. X86 programs are executed in a software simulator. Just as a weather or biological simulator creates an *in silico* model of a phenomenon, Chernoff's X86 simulator creates a "virtual" X86 and uses it to run X86 programs. As will be elaborated below, Chernoff maintains a strict wall of separation between his Alpha execution and his X86 execution – nothing will work right if, for example, Chernoff tries to manage an Alpha program using the X86 program execution manager, or vice versa.

The Office Action points to three totally unrelated portions of the Chernoff '028 patent. One section, col. 25 line 38 to col. 26 line 12, relates to multi-threading of X86 tasks, that is, changing from one thread to another. The second, at col. 88, lines 22-40, relates to exception handling, that is, correcting problems that arise during execution, so that the same program can resume execution. The third, at col. 33, lines 1-9, relates to subroutine calls, which generally only incidentally raise an exception, and do not cause any change of execution thread.

1. Background: Multi-Threading

A number of programming systems provide a feature called "multi-threading" (called "multitasking" of threads in Chernoff '028). From a user's point of view, multi-threading creates the appearance that a single program is doing multiple things simultaneously. As a rule of thumb

generalization, "multi-tasking" generally involves multiple programs, while "multi-threading" involves a single program. For example, most word processors allow a user to edit a document in the "foreground" while the document is being printed by the same program in the "background," but it is the same program executing in both threads. http://en.wikipedia.org/wiki/Thread_%28computer_science%29 (see Exhibit 3 to the paper of March 2005) gives an overview:

> Many programming languages, operating systems, and other software development environments support what are called "**threads**" of execution. Threads are similar to processes, in that both represent a single sequence of instructions executed in parallel with other sequences, either by time slicing or multiprocessing. Threads are a way for a program to split itself into two or more simultaneously running tasks. (The name "thread" is by analogy with the way that a number of threads are interwoven to make a piece of fabric).
>
> A common use of threads is having one thread paying attention to the graphical user interface, while others do a long calculation in the background. As a result, the application more readily responds to user's interaction.

The advantage of "threads" is that communications and switching between two threads of a single program is more efficient than communications and switching between two entirely different programs. Because the same program remains in execution, the amount of state that needs to be saved and restored is much smaller than in a full process context switch.[1] For example, a context switch among threads might be analogous to switching from reading one document to reading another document, both documents relating to the same matter. One document need only be laid down on the desktop, and another picked up, without the need to file the entire matter away.

The basic implementation techniques are a simplified version of multi-tasking discussed in § V.A, above: the state of one thread is saved to memory, the state of another thread is loaded from memory into execution, hardware, and control is transferred to the new thread.

Though threads are less tied to particular hardware than are processes, traditionally, it has been impossible to use a thread manager for one family of computers (Intel X86/Pentium, for example) to manage threads of a foreign computer (Digital Equipment Corp.'s Alpha, discussed

---

[1] The Wikipedia definition states that a context switch between processes is also a context switch between the threads of those processes.

in Chernoff '028, or the Tapestry processor discussed in this application's specification). As will be shown below, all of the multi-tasking and multi-threading managers discussed in Chernoff are specialized to either Alpha or X86 – Chernoff takes care that his Alpha programs are only managed by his Alpha program managers, never the X86 program manager, and vice-versa.

2. Background: Software Simulators and the Chernoff '028 Patent

Some manufacturers of computer hardware have found it valuable to allow their computers to run software that is coded in a foreign instruction set. For example, the Chernoff '028 patent discusses a software system from Digital Equipment Corp. that allows Digital's Alpha processor to run Intel X86 application programs, using a "simulator" or an "interpreter." An interpreter is software that creates a "virtual" CPU of the foreign operating system.

An interpreter also has to provide many or most of the features of an operating system. For example, X86 programs cannot use the multi-tasking features of the Alpha operating system, because the Alpha operating system understands only Alpha hardware, not X86 hardware.

In order to support multi-threading capabilities in Intel X86 application programs, Chernoff's X86-to-Alpha interpreter system must provide a multi-threading capability totally separate from the multi-tasking and multi-threading capabilities that can be used to manage programs written for the Alpha. The context-switch capability of the Alpha operating system cannot context switch among Intel X86 threads, because, as noted above, thread managers are tied to particular families of hardware.

Chernoff '028 discusses a totally separate layer of multi-threading, only for X86 processes, at col. 25, line 36 to col. 26, line 21. For example, Chernoff '028 at col. 25, lines 51-59 discusses saving and restoring the virtual X86 registers in a thread manager for X86 programs.[2] These are X86 hardware structures. Notably, all of the registers and structures, and "current state" mentioned in this section are X86 structures that are implemented as "virtual" software entities in the interpreter. This section of Chernoff '028 never discusses using the X86 multi-threading software to manage native Alpha processes, or switch between Alpha threads.

---

[2] "... the condition code bits [and] copies of the integer registers EAX 104a, EBX 104b, ECS 104c, EDX 104d EDI 104e, ESI 104f, EBP 104g and ESP 104h."

Alpha uses different hardware structures (Chernoff '028, col. 2, lines 60-63), and there is simply no way to access X86 registers in the Alpha instruction set, or vice-versa.

3. Background: Chernoff '028 and Exception Handling

At col. 88, lines 22-40, Chernoff '028 discusses exception handling. Certain kinds of events in a computer raise an "exception" or "interrupt," for example, divide-by-zero, an attempt to read or write an illegal memory location, a completion of a read or write to a disk, a timer expiration etc.

In most modern processors, the hardware classifies the exception into one of several categories and saves the state of the interrupted program. Then control is transferred to a "handler," software that further diagnoses the condition, and repairs it if possible. If the repair is successful, the handler may return control to the interrupted program, so that the program resumes as if the exception had never occurred. If no repair is possible, then the program is aborted.

Chernoff '028 at col. 88, lines 22-40 discusses how exceptions that arise in the execution of Alpha instructions are routed to a native Alpha handler for resolution (col. 88, line 28), and exceptions that arise in the context of the execution of the virtual X86 are routed to an X86 CISC exception handler (col. 88, line 47). These are necessarily entirely different. Alpha and X86 handle floating point exceptions entirely differently, and it simply would not do to handle an exception in an Alpha F-format or G-format floating-point number using the X86 handler.

Notably, this section of Chernoff '028 does not mention context switches. Nor does it mention handling an X86 exception in an Alpha handler, or vice-versa – at most, the mismatched handler determines that the exception must be handled elsewhere, and hands of the task of actually handling it. This is unsurprising: exception handlers and multi-threading handlers are completely different things.

C. The Embodiment in the Specification

With that background to establish some of the vocabulary, we can turn to the specification for this application. The specification discusses several techniques that allow the use of hardware and an operating system that are mismatched to each other.

Sections III and IV of the specification (pages 32-60) discuss mechanisms for allowing a non-native operating system, such as Microsoft Windows, to be used on hardware for which that operating system software has not been tailored, and to manage programs coded in the instruction set of that hardware. An introduction to this preferred embodiment appears at pp. 32-33 of the specification:[3]

> Referring to **Fig. 3a** and to Table 1, X86 threads (*e.g.*, **302**, **304**) managed by X86 operating system **306**, carry the normal X86 context, including the X86 registers, as represented in the low-order halves of r32-r55, the EFLAGS bits that affect execution of X86 instructions, the current segment registers, etc. In addition, if an X86 thread **302**, **304** calls native Tapestry libraries **308**, X86 thread **302**, **304** may embody a good deal of extended context, the portion of the Tapestry processor context beyond the content of the X86 architecture. A thread's extended context may include the various Tapestry processor registers, general registers r1-r31 and r56-r63, and the high-order halves of r32-r55 (see Table 1), the current value of ISA bit **194**...
>
> The Tapestry system manages an entire virtual X86 **310**, with all of its processes and threads, *e.g.*, **302**, **304**, as a single Tapestry process **311**. Tapestry operating system **312** can use conventional techniques for saving and restoring processor context, including ISA bit **194** of PSW **190**, on context switches between Tapestry processes **311**, **314**. However, for threads **302**, **304** managed by an off-the-shelf X86 operating system **306** (such as Microsoft Windows or IBM OS/2) within virtual X86 process **311**, the Tapestry system performs some additional housekeeping on entry and exit to virtual X86 **310**, in order to save and restore the extended context, and to maintain the association between extended context information and threads **302**, **304** managed by X86 operating system **306**. (Recall that Tapestry emulation manager **316** runs beneath X86 operating system **306**, and is therefore unaware of entities managed by X86 operating system **306**, such as processes and threads **302**, **304**.)
>
> **Figs. 3a-3n** describe the mechanism used to save and restore the full context of an X86 thread **304** (that is, a thread that is under management of X86 operating system **306**, and thus invisible to Tapestry operating system **312**) that is currently using Tapestry extended resources. In overview, this mechanism snapshots the full extended context into a memory location **355** that is architecturally invisible to virtual X86 **310**. A correspondence between the stored context memory location **355** and its X86 thread **304** is maintained by Tapestry operating system **312** and X86 emulator **316** in a manner that that does not require cooperation of X86 operating system **306**, so that the extended context will be restored when X86 operating system **306** resumes X86 thread **304**, even if X86

[3] This discussion, like all of § V, is a mere discussion of one concrete preferred embodiment, as a helpful aid to establish context for the discussion of the claims that follows. It is not a discussion of the claims themselves.

> operating system **306** performs several context switches ... before the interrupted X86 thread **304** resumes. The X86 emulator **316** or Tapestry operating system **312** briefly gains control at each transition from X86 to Tapestry or back, including entries to and returns from X86 operating system **306**, to save the extended context and restore it at the appropriate time.
>
> The particulars of the claimed subject matter are discussed in relation to specific claims.

VI. Standard of Review

On the merits, the Board reviews an Examiner's factual and legal conclusions without deference. The initial burden to come forward with evidence always remains with the examiner. No burden shifts to the applicant until the Examiner has addressed every element of a *prima facie* case of unpatentability. *E.g.*, MPEP § 2142.

As a matter of administrative law, agency decisions are void – that is, they have no legal existence – when an agency employee fails to make the showings required in the agency's own procedural handbook. *Service v. Dulles*, 354 U.S. 363, 388-89 (1957). As will be shown below, the only "rejection" in the entire Office Action that is not in direct violation of mandatory instructions of the Director is the § 101 issue, and even that is incorrect on the merits.

An agency action must be set aside if it is contrary to law, in excess of authority, short of statutory right, in excess of statutory jurisdiction, or without observance of procedure. 5 U.S.C. § 706(2). Review under these prongs of § 706 carries no deference whatsoever. *E.g.*, *Vitarelli v. Seaton*, 359 U.S. 535, 546-47 (1959) (Frankfurter, J., concurring) ("procedure must be scrupulously observed."); *Reuters v. F.C.C.*, 781 F.2d 946, 950-51 (D.C. Cir. 1986) ("*Ad hoc* departures from [an agency's] rules, even to achieve laudable aims, cannot be sanctioned"). Reviews on the merits are entirely non-deferential when agency or Administrative Procedure Act procedures have been violated at any stage of agency proceedings. *Stone v. Federal Deposit Insurance Corp.*, 179 F.3d 1368, 1376 (Fed. Cir. 1999) ("Our system is premised on the procedural fairness at each stage of [agency] proceedings. [A party before an agency] is entitled to a certain amount of due process rights at each stage and, when those rights are undermined, the [party] is entitled to relief."). The reason for non-deferential review follows from well-established principles of administrative law: an action taken in violation of an agency's own regulations is "illegal and of no effect," *Vitarelli*, 359 U.S. at 545; *IMS, P.C. v Alvarez*, 129 F.3d

618, 621 (D.C. Cir. 1997) (it is a "well-settled rule that an agency's failure to follow its own regulations is fatal to the deviant action"), and if no agency decision exists, then there is no agency action to which to give deference.

An agency decision must be set aside if it is "arbitrary [or] capricious" 5 U.S.C. § 706(2)(A). An agency action is arbitrary and capricious if it fails to consider relevant factors or makes an unexplained departure from past norms, *Motor Vehicle Mfrs. Ass'n, Inc. v. State Farm Mutual Auto. Ins. Co.*, 463 U.S. 19, 57 (1983) ("an agency changing its course must supply a reasoned analysis ..."); *Atchison Topeka & Santa Fe Rwy Co. v. Wichita Board of Trade*, 412 U.S. 800, 808 (1973). "Arbitrary and capricious" review is non-deferential when an agency has failed to address all relevant factors or all exceptions raised to prior intermediate actions, has acted inconsistently with its own precedent, or has violated its own procedures. 5 U.S.C. § 557(c); *see Motor Vehicle Mfr's Assn.*, 463 U.S. at 48 ("an agency must cogently explain why it has exercised its discretion in a given manner"); *Atchison*, 412 U.S. at 806-07. Agencies only receive deference when their written decisions reflect a *bona fide* effort to engage in rational decision making.

VII. Argument

A. Paragraph 4 of the Office Action - § 112 ¶ 2 Issues

1. Substantive Legal Standard for Definiteness

The legal standard for definiteness under § 112 ¶ 2 is met if there is any reasonable, unambiguous reading of a claim, *Exxon Research and Engineering Co. v. United States*, 265 F.3d 1371, 1375, 60 USPQ2d 1272, 1276 (Fed. Cir. 2001):

> We have not insisted that claims be plain on their face in order to avoid condemnation for indefiniteness; rather what we have asked is that the claims be amenable to construction, however difficult that task may be. If a claim is insolubly ambiguous... we have held the claim indefinite. If the meaning of the claim is discernible, even though the task may be formidable and the conclusion may be one over which reasonable persons will disagree, we have held the claim sufficiently clear to avoid invalidity on indefiniteness grounds.

If a claim can be read in one reasonable way that conforms to the usual rules for parsing English sentences, and an unreasonable way that disregards those usual rules, then the unreasonable reading is simply irrelevant, and is not a basis for rejecting a claim.

2. Procedural Legal Standard for Raising an Indefiniteness Rejection

MPEP § 2173.02 states the minimum requirement for any rejection whatsoever to exist under § 112 ¶ 2 (emphasis added):

> **2173.02 Clarity and Precision**
> The examiner's focus during examination of claims for compliance with the requirement for definiteness of 35 U.S.C. 112, second paragraph, is whether the claim meets the threshold requirements of clarity and precision, not whether more suitable language or modes of expression are available. ... he or she should allow claims which define the patentable subject matter with a reasonable degree of particularity and distinctness. Some latitude in the manner of expression and the aptness of terms should be permitted even though the claim language is not as precise as the examiner might desire. Examiners ... should not reject claims or insist on their own preferences if other modes of expression selected by applicants satisfy the statutory requirement.

Section 112, second paragraph, requires a "reasonableness" inquiry. For example, no rejection even exists where an artificially-constructed unreasonable or grammatically-incorrect reading of a claim is considered, while a reasonable and grammatically correct reading is ignored. MPEP § 2173.02 continues:

> If the language of the claim is such that a person of ordinary skill in the art could not interpret the metes and bounds of the claim so as to understand how to avoid infringement, a rejection of the claim under 35 U.S.C. 112, second paragraph, would be appropriate. ... However, if the language used by applicant satisfies the statutory requirements of 35 U.S.C. 112, second paragraph, but the examiner merely wants the applicant to improve the clarity or precision of the language used, the claim must not be rejected under 35 U.S.C. 112, second paragraph, rather, the examiner should suggest improved language to the applicant.

Again, the standard is simple: any reasonable interpretation of a claim is sufficient under § 112 ¶ 2. An examiner has no discretion at all to impose personal preference.

3. Paragraph 4(a): "thread that are"

Paragraph 4(a) of the Office Action reads as follows:

> a. In claim 1, "thread that are" (line 21) is indefinite because it is grammatically incorrect and it is not made explicitly clear in the claim language

whether there is a singular or plural amount of threads. Claims 5 and 57 are rejected for the same reasons.

The last portion of claim 1 recites as follows:

> the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system.

The claim is grammatically correct, and easily understood by using the normal rules of parsing English sentences and dividing them into their constituent clauses and prepositional phrases.

Claim 1 recites "resources of the computer ... that are beyond ..." There is proper singular/plural agreement between "resources" and "are."

The underlined instance of "with the thread" is a separate prepositional phrase. No word of the claim requires singular/plural agreement with this instance of "thread."

Substantively, the claim is definite.

Procedurally, the Office Action fails to make the showings that are necessary to raise a rejection. When an Office Action is totally silent on any showing that no "person of ordinary skill in the art could [] interpret the metes and bounds," as required by MPEP § 2173.02, no rejection exists. When an Office Action makes no allegation that the one of ordinary skill would have difficulty understanding the scope of the claim, no rejection exists.

4. Paragraph 4(b), first half: "extended context" and "modified context"

Paragraph 4(b) of the Office Action reads as follows:

> b. In claim 1, the term "extended context" (line 19) is indefinite because it is not made explicitly clear whether this term relates to the "modified context" (line 6) or if it introduces a new type of context.

Substantively, the claim is drafted using normal claim drafting rules: when different words are used, they refer to "new" things:

- Claim 1 explicitly defines that the "modified context" is the context modified by "the entry handler programmed to ... modify the thread context before delivering the modified context to the operating system."
- Claim 1 then explicitly defines the "extended context" as including "resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system."

- The two contexts may overlap entirely, somewhat, or not at all.

The two claim limitations relate to each to exactly the extent recited in the claim, no more and no less.

Procedurally, the claim is drafted to conform to the rule stated in MPEP § 2173.04 ("Breadth is not indefiniteness"), and the Office Action sets forth no reason to believe that the claim means anything other than exactly what it says, that any person of ordinary skill would have difficulty establishing the metes and bounds of this claim, or that § 2173.04 does not apply to this claim. The Office Action makes no allegation or showing that there is any ambiguity in this claim. The Office Action exceeds the limits on examiner discretion set by the Director in MPEP § 2173.04. No rejection exists.

5. Paragraph 4(b), second half

The second half of paragraph 4(b) of the Office Action reads as follows:

> In addition, the phrase "the thread beyond those resources whose association with the thread is maintained by the operating system is vague and indefinite because it is not made explicitly clear how a thread can be beyond a resource for those resources. Claims 5, 46, 54-55, 57 and 78 are rejected for the same reasons.

Substantively, the claim is easily understood by applying the ordinary rules for parsing English sentences. In context, the last portion of claim 1 reads as follows:

> the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system

The ordinary parsing of the claim language is apparent and clear. The "extended context" includes particular "resources of the computer associated with the thread." The particular resources of the "extended context" are those resources "that are beyond" certain other resources. Those other resources are "those resources whose association with the thread is maintained by the operating system." Stated in another way, the extended context includes resources that are not "those resources whose association with the thread is maintained by the operating system."

Procedurally, the Office Action presents no basis to disregard this ordinary parsing of claim 1. The Office Action is inadequate to raise any rejection whatsoever.

6. Paragraph 4(c): "without modifying a pre-existing operating system"

Paragraph 4(c) of the Office Action reads as follows:

> c. In claim 1, the terms "without modifying a pre-existing operating system" and "without modifying the operating system" are indefinite because it is not made explicitly clear whether modifying of the operating system constitutes modifying its code, modifying its configurations, modifying its functions, modifying associated data registers, etc. Claims 5 and 56 are rejected for the same reasons.

To establish context for this language, the relevant paragraph of claim 1 reads as follows:

> without modifying the operating system, establishing a resumption exception to be raised on each resumption from the operating system complementary to one of the specified entries, the resumption exception having an associated exit handler, the exit handler programmed to restore the context saved by a corresponding execution of the entry handler.

Procedurally, paragraph 4(c) only demonstrates that some situations unambiguously fall within the claim, and others unambiguously do not. It raises no situation that results in uncertainty or indefiniteness.

- "Modifying the code," and "modifying its functions" are unambiguously within of "modifying a pre-existing operating system."
- As the Office Action itself concedes, "modifying associated data registers" is a change to something "associated" with an operating system, not a change to the operating system itself. This type of modification is unambiguously outside "modifying a pre-existing operating system."
- There are a number of different ways that the "configurations" of an operating system could be modified. When considered case-by-case, none are unambiguous. All involve either modifying code or functions, or modifying something "associated" with the operating system. The former is unambiguously "modifying a pre-existing operating system," and the latter is unambiguously not.

The Office Action identifies no ambiguity in the claim; it merely notes that there are some cases unambiguously within the claim, and some that are unambiguously excluded. That is the very definition of "definiteness."

In any event, substantively, the claims are "reasonably precise," and "as precise as the subject matter allows," and thereby meets the requirements of MPEP § 2173.02. As a practical matter, the "establishing an entry exception," "establishing a resumption exception," and "maintaining an association between one of the threads and an extended context of the thread"

almost certainly require creation or modification of some data structures, possibly associated with the operating system. They almost certainly require some code outside the operating system. However, the claims unambiguously recite that there is no modification of the "operating system" or "thread scheduler" itself associated with the result recited in the rest of the claim. Thus, these situations unambiguously fall within the respective claims. The Office Action offers no suggestion or showing that one of ordinary skill could possibly read the claim any other way, or would have any difficulty ascertaining the meaning of the claim.

7. Paragraph 4(d): "returning control"

Paragraph 4(d) questions the "antecedent basis" for "returning control," even though neither the word "said" nor the word "the" is used.

MPEP § 2173.05(e) makes clear that "antecedent basis" is not an independent or automatic ground of rejection; there must be an additional showing, that the claim is unclear to one of ordinary skill:

> **2173.05(e) Lack of Antecedent Basis**
> A claim is indefinite when it contains words or phrases whose meaning is unclear. ... Obviously, however, the failure to provide explicit antecedent basis for terms does not always render a claim indefinite. If the scope of a claim would be reasonably ascertainable by those skilled in the art, then the claim is not indefinite. ... Inherent components of elements recited have antecedent basis in the recitation of the components themselves. ... [T]he limitation "the outer surface of said sphere" would not require an antecedent recitation that the sphere has an outer surface.

The Office action merely states "lacks antecedent basis." Without a showing that that lack means that "the scope of a claim would [not] be reasonably ascertainable by those skilled in the art," no rejection exists.

"Returning control" is a well-established term of art that requires no definition or antecedent basis to be understood by those of ordinary skill. For example, claim 6 of U.S. Patent No. 6,826,675 and claim 1 of U.S. Pat. No. 6,711,644 use "returning control" in almost the same way as these claims. "Returning control"[4] is used in literally hundreds of computer patents – it is a term of art that requires no antecedent basis.

[4] Or similar language like "return of control" or "control is returned."

Further, the existing claim language of claim 33, "scheduling concurrent threads of control by the operating system" provides antecedent basis for "control." Claim 79 recites "returning control to a caller of the service routine:" because those in the art inherently associate a "caller" with a "return," the "caller" inherently provides antecedent basis for the return. Both claims thus provide "antecedent basis" pursuant to MPEP § 2173.05(e).

In sum, the Office Action proposes no set of facts that would present any unreasonable difficulty in "understand[ing] how to avoid infringement" of any claim. The Office Action makes no attempt to show that the claim is unreasonably imprecise. Without that showing, the Office Action is too incomplete to raise any § 112 ¶ 2 rejection whatsoever.

B. Paragraph 5 of the Office Action - § 101 Issues

Paragraph 5 of the Office Action asserts that claim 5 is "directed to method steps which can be practiced mentally in conjunction with pen and paper. ... [It] is uncertain what performs each of the method steps. The examiner suggests [changing] to 'computer implemented methods' in the preamble."

The Examiner is wrong on the facts. Claim 5 recites "scheduling concurrent threads of control by a pre-existing thread scheduler of a computer." Claim 5 is "certain" that the method step is performed by "a pre-existing thread scheduler of a computer." The Office Action's allusion to "mentally [with] pen and paper" is groundless and legally irrelevant.

The Examiner is wrong on the law. Second, the Office Action suggests that recitation of a computer in the body of a claim may be ignored, and that the Examiner will only pay attention to the preamble. The law is exactly opposite the Examiner's view. *Contrast In re Walter*, 618 F.2d 758, 767-70, 205 USPQ 397, 409 (CCPA 1980) (claim not patentable where computer "signals" appears only in the preamble: "The specific end use recited in the preambles does not save the claims from the holding in *Flook*.") with *In re Johnson*, 589 F.2d 1070, 200 USPQ 199 (CCPA 1980) (a claim directed to essentially similar subject matter is patentable under § 101, because the "seismic trace" signal appears in the body of the claim).

No rejection is warranted.

C. Claims 1-4, 5-32, 82, 83

One fundamental error pervades all prior art issues raised in the Office Action: the Office Action picks out a few isolated words of the claim, compares them to isolated words in the references. The Office Action ignores much of the claims, and entirely ignores what the reference has to teach about the few isolated words the Office Action singles out.

Claim 5 is discussed in connection with Nilsen '665 and Chernoff '028 at paragraphs 6, 8, and 97-100 of the Office Action. Claim 5 recites as follows:

> 5. A method, comprising:
>
> scheduling concurrent threads of control by a pre-existing thread scheduler of a computer, each thread having an associated context, an association between a thread and a set of computer resources of the associated context being maintained by the thread scheduler; and
>
> without modifying the thread scheduler, maintaining an association between one of the threads and an extended context of the thread through a context change induced by the thread scheduler, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the thread scheduler.

Claim 5 recites an "extended context" that includes "resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the thread scheduler." The Office Action suggests that this might be met by Chernoff '082. The Office Action makes a number of errors.

1. Procedurally, the Office Action Fails to Raise any Rejection Under § 103

The procedural minimum for raising any rejection under § 103(a) is set forth at MPEP §§ 2141-2144.09. The basics are summarized at MPEP § 2143 (underline added)

> **2143 Basic Requirements of a *Prima Facie* Case of Obviousness**
>
> To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations.
>
> The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, not in applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991).

MPEP § 2142 clarifies that unless an Office Action establishes a relationship between every limitation of a claim and the prior art, unless it shows "motivation to combine" from the prior art (not an examiner's conjecture), unless it makes some showing of "reasonable expectation of success," no rejection exists. The Office Action fails to comply with the procedural minima set out in 37 C.F.R. § 1.104(c)(2) and MPEP § 2141-2144.09 necessary to raise a rejection. Until these minima have been met, no rejection even exists.

The first error in the Office Action is that it is incomplete. The Office Action picks and chooses a few isolated words out of the claims, and ignores the rest. The Office Action makes no attempt to show that that the identified bits and pieces of the references are interconnected in the manner recited in the claims. For example, claim 1 recites an "operating system." The discussion of claim 1 (¶ 6 of the Office Action) simply ignores this claim limitation. As a second example, claim 5 recites a single thread scheduler in the first and second paragraphs of claim 5. The Office Action refers to two unrelated thread schedulers in Chernoff '028. Because the Examiner simply ignored so many words of the claims, he never recognized that Chernoff's two thread schedulers are not interrelated as recited in the claim. As a third example, claim 5 recites different "contexts" that are interrelated in certain ways – the Office Action simply ignores these interrelationships. MPEP § 2143.03 states that any § 103 rejection requires a showing that "all the claim limitations must be taught or suggested by the prior art." As a fourth example, the Office Action never identifies any resource in either reference that is a "resource beyond those resources whose association with the thread is maintained by the thread scheduler." The Office Action makes no attempt to comply with § 2143.03. No claim is rejected under § 103(a).

The second error is one of commission: the Office Action expressly admits, page 5, that neither reference teaches "maintaining an association between one of the threads and an extended context of the thread through a context change induced by the thread scheduler." The Office Action does not allege that this element is "capable of instant and unquestionable demonstration as being well-known." Instead, the Office Action argues that this claim limitation may be conjured out of thin air, based on nothing more than the Examiner's hope that this might be useful. The Director himself has instructed that this is absolutely impermissible. MPEP § 2143.03 ("All limitations must be taught or suggested..."). Federal Circuit precedent is

similar. *In re Zurko*, 258 F.3d 1379, 1385, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001) ("the Board cannot simply reach conclusions based on its own understanding or experience—or on its assessment of what would be basic knowledge or common sense. Rather, the Board must point to some concrete evidence in the record in support of these findings."); *In re Lee*, 277 F.3d 1338, 1343-44, 61 USPQ2d 1430, 1434 (Fed. Cir. 2002) ("'common knowledge and common sense' on which the Board relied in rejecting Lee's application are not the specialized knowledge and expertise contemplated by the Administrative Procedure Act. Conclusory statements such as those here provided do not fulfill the agency's obligation..."); *In re Blecher*, 991 F.2d 810 (table), 1993 WL 73726 at **3, 1993 U.S. App. Lexis 5423 at *8 (Fed. Cir. 1993) ("The PTO argues that common knowledge and common sense would have motivated the skilled artisan to ... However, in order for a conclusion of obviousness to be made from the common knowledge of the skilled artisan, it must be established that 'this knowledge was in the art.'"). Without some citation to a permissible class of evidence, the Office Action fails to comply with MPEP § 2143.03, and no rejection exists.[5]

Third, the Office Action asserts that Nilsen's handlers would find some unspecified benefit in having control over Chernoff's "context data structure," but cites no prior art to support this statement. Statements of "motivation to combine" must be supported by substantial evidence.[6] Applicant requests a reference to support the "motivation to combine" under 37 C.F.R. § 1.104(d)(2).

---

[5] To the degree that "maintaining an association between one of the threads and an extended context of the thread through a context change induced by the thread scheduler" and having control over Chernoff's "context data structure" involve official notice, applicant calls for a reference or an affidavit pursuant to 37 C.F.R. § 1.104(d)(2).

[6] MPEP § 2143.03 restates Federal Circuit law: *every* limitation of a claim must be met by prior art drawn from one of the categories of § 102. No limitation may be rejected based on bald assertion. *In re Zurko*, 258 F.3d 1379, 1385, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001) ("the Board cannot simply reach conclusions based on its own understanding or experience—or on its assessment of what would be basic knowledge or common sense. Rather, the Board must point to some concrete evidence in the record in support of these findings."); *In re Lee*, 277 F.3d 1338, 1343-44, 61 USPQ2d 1430, 1434 (Fed. Cir. 2002) ("'common knowledge and common sense' on which the Board relied in rejecting Lee's application are not the specialized knowledge and expertise contemplated by the Administrative Procedure Act. Conclusory statements such as those here provided do not fulfill the agency's obligation."); *Motorola v. Interdigital Technology Corp.*, 121 F.3d 1461, 1466-67, 43 USPQ2d 1490, 1490-91 (Fed. Cir. 1997)

Fourth, the Office Action makes no showing that the three distantly-separated and logically unrelated portions of Chernoff '028 have any relationship to each other, and makes no showing of "motivation to combine" them. Because the Office Action fails to comply with MPEP § 2143.01, it fails to raise a rejection.

Fifth, the Office Action is totally silent on "reasonable expectation of success," as required by MPEP § 2143.02. Paragraph 98 of the January 2005 Office Action confuses two entirely separate issues: the MPEP test for obviousness states that "motivation to combine" and "reasonable expectation of success" are two distinct elements – showing one does not constitute a showing of the other. MPEP §§ 2143, 2143.01 and 2143.02.

Without the showings required by MPEP §§ 2143.01 and 2143.02, no *prima facie* rejection exists.

These first five errors demonstrate that the Office Action does not comply with agency rules. As such, the Supreme Court states that the Office Action is "illegal and of no effect." *Vitarelli*, 359 U.S. at 545. There is no rejection to affirm.

2. Claims 1-4, 5-32, 82 and 83 are Patentable on the Merits

a) a "thread scheduler" to perform a "context change" using an "extended context" that is "beyond" the "thread scheduler"

The first paragraph of claim 5 recites a "context" and a "pre-existing thread scheduler" that are designed to work with each other. The second paragraph of claim 5 then recites an "extended context" that is "beyond" the capabilities of the "thread scheduler." Claim 5 recites that, even though the "thread scheduler" is unable to manage the "extended context," nonetheless the scheduler and extended context are made to work with each other by "maintaining an association" between the "extended context" and a thread that can be scheduled by the "pre-existing thread scheduler." In some embodiments, claim 5 might allow the use of a mature operating system (*e.g.*, Microsoft Windows) designed for one computer architecture to be used with a computer of an entirely different architecture.

(every element must be met by prior art, even elements that are "well known" standing alone). If DiBrino is not offered as a "new ground of rejection," then no rejection exists at all.

The Examiner' sixth error appears in paragraph 97, which responds to this argument by equating "thread schedulers" with "handlers." Chernoff '028 makes clear that the two things are quite different. A "thread scheduler" is usually designed to restore a thread to execution in exactly the same state[7] from which the thread was suspended; in contrast, a handler is almost always designed to change the thread's state by removing the condition that raised the exception.

The three portions of Chernoff '028 indicated in the Office Action do not relate to each other, and cannot be combined to meet the claim language. For example, col. 25, line 4 to col. 26 line 12 discusses a non-native "context data structure" 180, but only discusses software that is designed to work with that very "context data structure," not an "extended context" that is "beyond" that software. The Office Action then jumps 60 columns away to col. 88, lines 22-40. But col. 88 never mentions the "context data structure 180" or its associated software – instead, it discusses an unrelated topic. Col. 88 discusses a fairly conventional exception mechanism that stores the context of a program in the conventional way, dispatches to a handler through a dispatch table in the conventional way, and handles the exception in the conventional way. This portion of Chernoff '028 does not mention any problem that would call for an "extended context," let alone suggest the use of an "extended context" that corresponds to claim 5.

The January 2005 Office Action raises a new ground of rejection, relating to col. 33, lines 1-9 of Chernoff '028. Col. 33 relates to a third unrelated topic, even further afield from the claim. This portion relates to simple subroutine calls, in this case from a portion of the program that has been binary translated from X86 to Alpha instructions (col. 32, lines 63-64) to one that is still in pure X86 code to be executed in the interpreter. Typical subroutine calls do not cause an interrupt, a context save, or a context switch. Nothing in this portion of Chernoff '028 suggests anything atypical in any of these respects that would suggest any interaction with the mechanisms of col. 25-26 or col. 88.

The seventh error is a misreading of the references, and a mistaken usage of several terms of art. At the end of paragraph 6, the Office Action states "It would have been obvious to ... because it allows the handlers to have control over the context data structure stored in the table."

---

[7] Or, possibly, as modified by something other than the thread scheduler itself.

The Examiners' paraphrase of the reference is simply wrong. Chernoff's "table" stores "pointers to methods," Chernoff '028, claim 1, and col. 3, line 55, not to the "context data structures."

The Office Action itself concedes that Nilsen '665 teaches nothing corresponding to the "extended context" of claim 5.

Because the examiner cannot point to either reference to support allowing a "thread scheduler" to perform a "context change" using an "extended context" that is "beyond" the "thread scheduler," claim 5 is patentable.

b) "Motivation to Combine"

In an eighth error, the Office Action pieces together three unrelated and incompatible portions of Chernoff '028 to attempt to meet the "operating system" of claim 1 or "thread scheduler" of claim 5. This is apparently based in the Examiner's technological misunderstanding: the Office Action fails to recognize that Chernoff '028 manages his X86 threads and his Alpha threads in two completely separate ways, and never mixes the two. The Office Action focuses most on col. 25, line 38 to col. 26, line 12 of Chernoff '082, discussing the "Context Data Structure" 180. Chernoff '028 teaches that his context data structure holds only X86 contexts (col. 25, lines 50-65). Chernoff '028 only discusses using this X86 multitasking manager to manage X86 processes (col. 25, lines 41-42). Thus, Chernoff's multitasking manager and context data structure discussed here are exactly tailored to each other – the manager only manages X86 context data, and the context data structure only holds X86 context data. Chernoff's "context data structure" never stores a "resource [for example, an Alpha register] beyond those resources whose association with the thread is maintained by" Chernoff's multitasking manager of col. 25.

Alpha context is not mentioned in this portion of Chernoff '028. Indeed, it would be quite impossible to store or switch Alpha context using the mechanisms described in col. 25-26. For example, the Alpha processor has 31 integer registers and 31 floating-point registers that must be stored on context switch, each 64 bits wide (see the "Compaq Computer Corp., Compiler Writer's Guide for the Alpha 21264 (1999)," at pages 3-9 to 3-11, of record in this application, or at `//ftp.digital.com/pub/Digital/info/semiconductor/literature/cmpwrgd.pdf`). Alpha's data registers alone require almost 4000 bits of storage. In contrast,

Chernoff '028 teaches that his X86 "context data structure" only has space for the X86's eight 32-bit integer registers (col. 25, lines 56-59; col. 16, lines 32-33), a total of only 256 bits. 4000 bits of information cannot possibly fit in 256 bits of storage.[8]

The Office Action then brings in other unrelated portions of Chernoff '028. These portions are unrelated to each other, to col. 25-26, or to the "extended context" recited in claim 5. For example, col. 88, lines 22-40 never mentions the "context data structure 180" or its associated multitasking manager. Instead, col. 88 discusses a fairly conventional exception mechanism that stores Alpha context of an Alpha program in the conventional way (unrelated to the X86 context save discussed in col. 25-26), dispatches to an Alpha handler, and handles Alpha exceptions in the conventional way. Even the residual X86 case, col. 88, lines 45-52, has no relationship to the X86 mechanisms discussed in col. 25-26. The Office Action does not suggest that col. 88 of Chernoff '028 has any relationship to the other portions of Chernoff, or to the "extended context" recited in claim 5, and none is apparent.

No single component of Chernoff '028 meets all the limitations and interrelationships of the "extended context" of claim 5. The Office Action makes no attempt to show such a correspondence, and none exists. No rejection exists, and none could be based on Nilsen '665 and Chernoff '028.

Because claim 5 recites a feature that is absent from both Nilsen '665 and Chernoff '028, claim 5 is patentable over this combination.

3. Claims 1-4, 6-32, 46, 54, 57, 58, 76-78, 82, and 83 are patentable with Claim 5

Claim 1 recites similar language, a "pre-existing operating system" that works with "extended context" in a similar way. Claim 1 is patentable for similar reasons.

Claims 2-4, 6-32, 46, 54, 57, 58, 76-78, 82, and 83 are dependent on claims 1 or 5, or recite language similar to that discussed above, and patentable therewith.

---

[8] Both processors have a number of control registers that must also be stored; Alpha's set is larger than the storage provided for them in Chernoff '028.

D. Claims 33-55 and 56-78

Claim 33 is mentioned in paragraphs 6 and 25 of the Office Action. Claim 33 recites as follows (though may be amended, without altering its scope, if the Examiner opts to enter the Proposed Amendment filed March 21, 2005):

> 33. A method, comprising:
>
> establishing an entry exception to be raised on each entry to a computer operating system at a specified entry point or on a specified condition;
>
> establishing a resumption exception to be raised on each resumption from the operating system complementary to one of the specified entries;
>
> on detecting a specified entry to the operating system from an interrupted process of the computer, raising and servicing the entry exception, and then entering the operating system to perform a service associated with the original operating system entry; and
>
> on detecting a complementary resumption, raising and servicing the resumption exception, and returning control to the interrupted process.

1. As a Matter of Administrative Procedure Law, These Claims Are Not Rejected

For the same reasons discussed above in § VII.C.1, the Office Action is procedurally inadequate to raise any rejection whatsoever.

Neither the June 2004 Office Action nor the January 2005 office Action identify particular features of Nilsen '665 that might correspond to the particular exceptions recited in these claims, or make any comparison of these claim limitations to Chernoff '028. The January 2005 Office Action merely states that "Chernoff and Nilsen both teach using exception and interrupt handling to control the execution of instructions." No claim recites the language discussed in the Office Action, and the Office Action does not address the language recited in the claims. Thus no rejection exists.

Paragraphs 6, 25 and 101 of the Office Action assert that Nilsen '665 teaches "an entry exception," and an "exit exception" at col. 25, lines 42-67. This is not correct. This very topic was discussed in the Interview of July 2, 2003. The Summary of that Interview (memorialized at Applicant's Response of July 3, 2003, at page 5) reads as follows (underline added):

> In the interview, it was agreed that col. 25, lines 40-67, col. 26, lines 6-15, col. 33, lines 40-67 and col. 37, lines 60-67 of the Nilsen '665 patent do not discuss any feature analogous to the "entry exception" and "exit exception" recited in claim 33. Further, Nilsen '667 makes only minor mention of the "operating system,"

> and never in a context that relates to the "operating system" limitations of the claim.

The Examiner attempts to walk away from this agreement that was properly memorialized. See MPEP § 713.04. Applicant is unaware of any procedure or rule that permits him to retroactively withdraw from an agreement, without providing a detailed explanation of his revised position.

The January 2005 does not identify particular features of Nilsen '665 that might correspond to these exceptions, or make any comparison whatsoever of these claim limitations to Chernoff '028. The Office Action makes no showing of "motivation to combine" or "reasonable expectation of success." Thus no rejection exists.

Independent claim 56 recites similar language. For similar reasons, no rejection of claim 56 exists.

2. Claims 33-55 and 56-78 Are Patentable On the Merits

Claim 33 recites wrapping a conventional or pre-existing entry point of a computer operating system in a pair of additional exceptions: one to be raised on entry, one to be raised on resumption. One example embodiment can be seen in Fig. 3a of this application, by following arrows (7), (8), (9), (12), (13) and (14). In this example, when an interrupt occurs (388 of Fig. 3a), normally execution would trap into the operating system (306 of Fig. 3a). Instead, a second exception (arrow (7) of Fig. 3a, corresponding to the "entry exception" of claim 33) is raised. This causes control to be transferred to a handler (350 of Fig. 3a, corresponding to "servicing the entry exception" of claim 33). When that handler returns control (arrow (9) of Fig. 3a), typically the operating system handles the original interrupt. When the operating system executes its return of control instruction (arrow (12) of Fig. 3a, corresponding to "a complementary resumption" of claim 33), another exception (arrow (13) of Fig. 3a, corresponding to the "resumption exception" of claim 33) is raised. This exception transfers control to a third handler (lower right corner of Fig. 3a, "servicing the resumption exception" of claim 33). The resumption exception handler returns control to the original thread (arrow (14) of Fig. 3a, corresponding to the "returning control to the interrupted process" of claim 33). Thus, a typical

embodiment of claim 33 would involve three exceptions and associated handlers. See Figs. 3h, 3i and 3j.[9]

Paragraph 6 of the Office Action juxtaposes the "resumption exception to be raised on each resumption from the operating system complementary to one of the specified entries" to Nilsen '665, col. 30, lines 25-28, col. 33, lines 40-67, col. 37, lines 60-67, and col. 25, lines 40-67). The Office Action merely designates these portions of Nilsen '665, without identifying any "resumption exception" being raised there, or otherwise explaining the pertinence.[10] These portions of Nilsen '665 discuss nothing corresponding to the "resumption exception." The designated portions at best discuss an implementation of the setjmp/longjmp feature of the standard "C" library, for handling a single exception (See Exhibit 4 to the paper of March 2005). These portions of Nilsen '665 never suggest that a "resumption exception" is raised, separate from the condition that triggered the setjmp/longjmp itself. Any particular use of setjmp/longjmp involves only one exception and one handler, and nothing corresponding to the "resumption exception" following the setjmp/longjmp, as required by claim 33.[11]

The Office Action does not suggest that Chernoff '028 teaches anything corresponding to the "resumption exception," and a brief review of Chernoff '028 reveals no such teaching.

Claim 33 recites a limitation absent from all references. Claim 33 is therefore patentable. Independent claims 1 and 56 recites similar language and are patentable for similar reasons. Dependent claims 2-4, 6-14, 34-55, and 57-78 either recite similar language, or are dependent on these claims, and are therefore patentable.

---

[9] This comparison of the specification to the claim is not a limiting description of the invention or the claim; it is merely offered as a concrete preferred embodiment to assist in understanding the claim.

[10] Because the Office Action fails to comply with the requirements of 37 C.F.R. § 1.104(b)(2) for setting out a rejection, no rejection has been raised.

[11] MPEP § 713.04 provides no basis for giving an applicant's interview summary less weight than the examiner's summary. Applicant's paper of July 3, 2003 was filed one day after the interview it records. The Examiner's remarks of January 2005 are 18 months after the fact. Applicant's interview summary is therefore entitled to weight as an accurate recordation of the conversation.

E. Claim 79

Claim 79 is discussed in paragraphs 6, 55 and 102 of the Office Action. Claim 79 recites as follows:

> 79. A method, comprising:
>
> during invocation of a service routine of a computer, passing a linkage return address to the service routine at which to resume execution on completion of the service, the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception;
>
> on return from the service routine, attempting to execute the instruction at the linkage return address and raising the chosen exception; and
>
> after servicing the exception, returning control to a caller of the service routine.

Claim 79 recites "the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception." One example is discussed at § IV.H, at page 49-50 of the specification.

1. Claim 79 Is Not Rejected

For reasons analogous to those discussed above in § VII.C.1, the Office Action is procedurally inadequate to raise any rejection whatsoever.

Paragraph 55 of the Office Action asserts that claim 79 is "rejected for the same reasons stated in the rejection of claims 1 and 33." This cannot be so. Neither claim 1 nor claim 33 recite language similar to the language underlined above. This sentence of the Office Action raises no rejection.

Paragraph 57 then admits that nothing in either Nilsen '665 or Chernoff '028 corresponds to the underlined language. Paragraph 57 nonetheless asserts that the language would be obvious. The Examiner's attention is drawn to MPEP § 2143.03:

> **2143.03 All Claim Limitations Must Be Taught or Suggested**
>
> To establish *prima facie* obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). ...

The Office Action identifies no point in either Nilsen '665 or Chernoff '028 that even "suggests" an "address ... deliberately chosen [to] raise a program execution exception." No rejection exists.

The Examiner then asserts, with no citation to any prior art, that a technique unknown in the art is nonetheless obvious. In MPEP § 2143.03, the Director explicitly forbade this line of reasoning. The Office Action exceeds the discretion granted to examiners by the Director. It raises no rejection.

Further, MPEP § 2143.03 is based on Federal Circuit law, which holds that there is no obviousness once a claim element is admitted to be totally novel to the art. To render a claim obvious, the relied-upon prior art must teach every element of the claim. Even if the missing element is well-known in the art, specific prior art must be put forward against the limitation. *E.g.*, *Motorola v. Interdigital Technology Corp.*, 121 F.3d 1461, 1466-67, 43 USPQ2d 1490, 1490-91 (Fed. Cir. 1997). If any future rejection maintains that an element totally absent from the art is nonetheless obvious, Applicant requests written authority that overrules MPEP § 2143.03 and *Motorola*.

2. Claim 79 Is Patentable On the Merits

The Office Action itself admits that that nothing in either Nilsen '665 or Chernoff '028 corresponds to the underlined language. Paragraph 102 of the January 2005 Office Action raises a new ground of rejection, attempting to fill the hole with the Abstract of DiBrino '894.[12] But DiBrino's abstract demonstrates the absence of the underlined claim language from the prior art. For example, DiBrino's abstract never mentions a "return from a service routine."[13] This limitation is absent from the prior art, and the claim therefore cannot be obvious. MPEP § 2143.03.

---

[12] See footnote 6.

[13] Further reliance on "official notice" is impermissible for two reasons. First, MPEP § 2144.03(A) cautions "It would not be appropriate for the examiner to take official notice of facts without citing a prior art reference where the facts asserted to be well known are not capable of instant and unquestionable demonstration as being well-known." The clear failure of DiBrino to supply all of the "official notice" subject matter shows that the original assertion of "official notice was incorrect." Second, "It is never appropriate to rely solely on 'common knowledge' in the art without evidentiary support in the record, as the principal evidence upon which a rejection was based."

The Office Action states that the proposed combination would "provide for control for exceptions to occur," but identifies no respect in which the control over exceptions in Nilsen '665 or Chernoff '028 is inadequate, how the supposed control would be exercised, or any respect in which the combination Nilsen '665 and Chernoff '028 offers any benefit over the two standing separately. A showing of "motivation to modify" must be more than "statements of generalized advantages and convenient assumptions." *In re Beasley*, 117 Fed. Appx. 739, 744 (Fed. Cir. 2004). No credible motivation to modify or combine is shown.

To the extent that the current Office Action relies on "common knowledge" or "well-known prior art," pursuant to 37 C.F.R. § 1.104(d)(2), Applicant calls for a reference or an affidavit showing "that the linkage return address is being deliberately chosen to that an exception is raised." Applicant calls for a reference that shows the entire phrase that the examiner asserts to be well-known, not a few isolated words.

For these reasons, claim 79 is patentable over the art. Claims 80-81 are dependent thereon, and allowable therewith. Claims 52-53 recite similar language, and are patentable for similar reasons.

VIII. Conclusion

In view of these remarks, Applicant respectfully submits that the claims are in condition for allowance. Applicant requests that the application be passed to issue in due course. In the event that any § 112 ¶ 2 rejection is thought to apply, Applicant requests entry of the amendments proposed in March 2005.

In the event any rejection is thought to apply, Applicant again requests an interview. Supervisory Examiner An indicated in the telephone interview of July 7 that an interview is proper at this time.

The Examiner is urged to telephone Applicant's undersigned counsel at the number noted below if it will advance the prosecution of this application, or with any suggestion to resolve any condition that would impede allowance. In the event that any extension of time is required, Applicant petitions for that extension of time required to make this response timely. Kindly

charge any additional fee, or credit any surplus, to Deposit Account No. 23-2405, Order No. 114596-05-4013.

Respectfully submitted,

WILLKIE FARR & GALLAGHER LLP

Dated: July 13, 2005

By: _____

David E. Boundy
Registration No. 36,461

WILLKIE FARR & GALLAGHER LLP
787 Seventh Ave.
New York, New York 10019
(212) 728-8000
(212) 728-8111 Fax

Claims Appendix to

Preliminary Comments Accompanying Notice of Appeal

CLAIMS

1. (last amended 9/13/04) A method, comprising:

without modifying a pre-existing operating system of the computer, establishing an entry exception to be raised on each entry to the operating system at a specified entry point or on a specified condition, the entry exception having an associated entry handler, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified context to the operating system;

without modifying the operating system, establishing a resumption exception to be raised on each resumption from the operating system complementary to one of the specified entries, the resumption exception having an associated exit handler, the exit handler programmed to restore the context saved by a corresponding execution of the entry handler;

scheduling concurrent threads of control by the operating system, each thread having an associated context, the association between a thread and a set of computer resources of the associated context being maintained by the operating system;

on detecting a specified entry to the operating system from an interrupted thread of the computer, raising and servicing the entry exception; and

on detecting a complementary resumption, raising and servicing the resumption exception, and returning control to the interrupted thread;

the entry exception, resumption exception, entry handler, and exit handler being cooperatively designed to maintain an association between one of the threads and an extended context of the thread through a context change induced by the operating system, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system.

2. (original) The method of claim 1, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

3. (original) The method of claim 1, wherein the operating-system-maintained resources of the thread context include data registers of the non-native computer architecture, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the non-native operating system.

4. (original) The method of claim 1, wherein thread scheduler and the thread execute in different instruction sets of the computer, and the entry and exit exception are automatically invoked, without explicit software request, on a transition between the thread instruction set and the operating system instruction set.

5. (last amended 9/13/04) A method, comprising:

scheduling concurrent threads of control by a pre-existing thread scheduler of a computer, each thread having an associated context, an association between a thread and a set of computer resources of the associated context being maintained by the thread scheduler; and

without modifying the thread scheduler, maintaining an association between one of the threads and an extended context of the thread through a context change induced by the thread scheduler, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the thread scheduler.

6. (original) The method of claim 5, wherein the thread scheduler is a component of an operating system of the computer, and further comprising:

establishing an entry exception to be raised on each entry to the operating system at a specified entry point or on a specified condition;

establishing a resumption exception to be raised on a resumption from the operating system following on a specified entry;

on detecting a specified entry to the operating system from an interrupted process of the computer, raising the entry exception, and establishing the association as part of servicing the entry exception; and

raising the resumption exception, and as part of servicing the resumption exception, reestablishing the context in association with the resumed thread returning control to the interrupted process.

7. (original) The method of claim 6, wherein an exception handler for the entry exception is programmed to save a context of the interrupted process and modify the thread context before delivering the modified context to the operating system; and

an exception handler for the resumption exception is programmed to restore the context saved by a corresponding execution of the entry exception handler.

8. (original) The method of claim 7, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

9. (original) The method of claim 8, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

10. (original) The method of claim 8, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

11. (original) The method of claim 6, wherein the operating system is in a binary code for a computer architecture non-native to the architecture of the computer.

12. (original) The method of claim 11, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

13. (original) The method of claim 11, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

14. (original) The method of claim 11, wherein the resources of the context maintained in association with the thread by the non-native operating system include data registers of the non-native computer architecture, the method further comprising:

in the entry exception handler, modifying at least half of the data registers of the portion of the process context maintained by the non-native operating system before delivering the process to the non-native operating system, at least some of the modified registers being redundantly written with data to enable checking of the validity of the contents of the context in the resumption exception handler.

15. (original) The method of claim 6, wherein thread scheduler and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the thread scheduler execution mode.

16. (original) The method of claim 15, wherein the thread execution mode and the thread scheduler execution mode are two different instruction set architectures of the computer.

17. (original) The method of claim 6, further comprising:

during servicing the entry exception, saving a portion of the context of the computer, and altering the context of an interrupted thread before delivering the interrupted thread and its corresponding context to the operating system.

18. (original) The method of claim 6, further comprising the step of modifying a linkage return address for resumption of the thread to include information used to maintain the association.

19. (original) The method of claim 18, wherein the modification leaves at least half of the bits of the linkage return address intact.

20. (original) The method of claim 5, wherein the thread scheduler is an operating system for a computer architecture other than the architecture native to the computer.

21. (original) The method of claim 20, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

22. (original) The method of claim 20, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

23. (original) The method of claim 5, wherein thread scheduler and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the thread scheduler execution mode.

24. (original) The method of claim 23, wherein the thread execution mode and the thread scheduler execution mode are two different instruction set architectures of the computer.

25. (original) The method of claim 23, further comprising the step of setting of a register to a value that specifies actions to be taken by an exception handler invoked on the transition to convert operands from one form to another to conform to a data storage convention of the thread scheduler execution mode.

26. (original) The method of claim 5, further comprising:

in an interrupt handler of the computer, saving a portion of the context of the computer, and altering the context of an interrupted thread before delivering the interrupted thread and its corresponding context to the thread scheduler.

27. (original) The method of claim 20, wherein the operating-system-maintained resources of the thread context include data registers of the non-native computer architecture, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the non-native operating system.

28. (original) The method of claim 27, wherein at least some of the modified registers are overwritten by a timestamp.

29. (original) The method of claim 27, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

30. (original) The method of claim 5, further comprising the step of modifying a linkage return address for the thread to include information used to maintain the association.

31. (original) The method of claim 30, wherein the modification leaves at least half of the bits of the linkage return address intact.

32. (last amended 9/13/04) The method of claim 5, further comprising either the step of deferring delivery of an interrupt before interrupting the thread by a time sufficient to allow the thread to reach a checkpoint, or the step of rolling execution of the thread back to a checkpoint, the checkpoints being points in the execution of the thread where the amount of extended context, being the resources of the thread that are beyond those whose resource association with the thread is maintained by the thread scheduler, is reduced.

33. (original) A method, comprising:

establishing an entry exception to be raised on each entry to a computer operating system at a specified entry point or on a specified condition;

establishing a resumption exception to be raised on each resumption from the operating system complementary to one of the specified entries;

on detecting a specified entry to the operating system from an interrupted process of the computer, raising and servicing the entry exception, and then entering the operating system to perform a service associated with the original operating system entry; and

on detecting a complementary resumption, raising and servicing the resumption exception, and returning control to the interrupted process.

34. (original) The method of claim 33, wherein an exception handler for the entry exception is programmed to save a context of the interrupted process and modify the thread context before delivering the modified context to the operating system; and

an exception handler for the resumption exception is programmed to restore the context saved by a corresponding execution of the entry exception handler.

35. (original) The method of claim 34, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

36. (original) The method of claim 35, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

37. (original) The method of claim 35, wherein the resources of the context maintained in association with the thread by the operating system include data registers of the non-native computer architecture, the method further comprising:

in the entry exception handler, modifying at least half of the data registers of the portion of the process context maintained by the operating system before delivering the process to the non-native operating system, at least some of the modified registers being redundantly written with data to enable checking of the validity of the contents of the context in the resumption exception handler.

38. (original) The method of claim 34, wherein the operating system and the process execute in two different instruction set architectures of the computer, and at least some of the steps to maintain the association between the process and the context are automatically invoked, without explicit software request, on a transition between the instruction set architectures.

39. (original) The method of claim 34, further comprising the step of modifying a linkage return address for the process to include information used to restore the context.

40. (original) The method of claim 33, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer, unmodified for execution on the computer.

41. (original) The method of claim 40, wherein the computer additionally executes an operating system native to the computer, and each exception is classified for handling by one of the two operating systems.

42. (original) The method of claim 40, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

43. (original) The method of claim 33, wherein the operating system and the process execute in different execution modes of the computer, and the steps to maintain the association between the process and the context are automatically invoked, without explicit software request, on a transition between the process execution mode and the operating system execution mode.

44. (original) The method of claim 43, wherein the process execution mode and the operating system execution mode are two different instruction set architectures of the computer.

45. (original) The method of claim 33, wherein a service routine for the entry exception modifies at least half of the data registers of the portion of a process context maintained in association with the process by the operating system before delivering the process to the non-native operating system.

46. (last amended 9/13/04) The method of claim 45, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the extended context, the extended context being the resources beyond those whose resource association with the process is maintained by the operating system, is saved before the modifying.

47. (original) The method of claim 46, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the context.

48. (original) The method of claim 45, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the context.

49. (original) The method of claim 45, wherein at least some of the modified registers are overwritten by a timestamp.

50. (original) The method of claim 33, further comprising the step of modifying a linkage return address for the process to include information used to maintain the association.

51. (original) The method of claim 50, wherein the modification leaves at least half of the bits of the linkage return address intact.

52. (original) The method of claim 33, further comprising:

as part of servicing the entry exception, modifying a linkage return address of the interrupted process, the return address being deliberately chosen so that an attempt to execute an instruction from the return address on return from the operating system will raise the resumption exception.

53. (original) The method of claim 52, wherein the linkage return address is selected to point to a memory page having a memory attribute that raises the chosen exception on at attempt to execute an instruction from the page.

54. (last amended 9/13/04) The method of claim 33, further comprising either the step of rolling execution of the process back to a checkpoint in the execution of the process where the amount of extended context, the extended context being the resources of the

process context beyond those whose resource association with the process is maintained by the operating system, is reduced.

55. (last amended 9/13/04) The method of claim 33, further comprising either the step of deferring delivery of an interrupt before interrupting the process by a time sufficient to allow the process to reach a checkpoint in the execution of the process where the amount of extended context, the extended context being the resources of the process context beyond those whose resource association with the process is maintained by the operating system, is reduced.

56. (original) A method, comprising:

without modifying a pre-existing operating system of the computer, establishing an entry handler for execution at a specified entry point or on a specified entry condition to the operating system, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified context to the operating system;

without modifying the operating system, establishing an exit handler for execution on resumption from the operating system following an entry through the entry handler, the exit handler programmed to restore the context saved by a corresponding execution of the entry handler.

57. (last amended 9/13/04) The method of claim 56, further comprising:

scheduling concurrent threads of control by the operating system, each thread having an associated context, an association between a thread and a set of computer resources of the associated context being maintained by the operating system; and

the entry and exit handlers being programmed to maintain an association between one of the threads and an extended context of the thread through a context change induced by the operating system, the extended context including resources of the computer associated with the thread that are beyond those resources whose association with the thread is maintained by the operating system.

58. (original) The method of claim 57, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

59. (original) The method of claim 57, wherein the operating system and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the operating system execution mode.

60. (original) The method of claim 57, further comprising:

in the entry handler, saving a portion of the context of the computer, and altering the context of the interrupted thread before delivering the interrupted thread and its corresponding context to the operating system.

61. (original) The method of claim 57, wherein the entry handler alters at least half of the data registers of the portion of a thread context maintained in association with the thread by the operating system before delivering the thread to the operating system.

62. (original) The method of claim 57, further comprising the step of modifying a linkage return address for the thread to include information used to maintain the association.

63. (original) The method of claim 56, wherein the operating system is an operating system for a computer architecture other than the architecture native to the computer.

64. (original) The method of claim 63, wherein the computer additionally executes an operating system native to the computer, and each interrupt or exception is classified for handling by one of the two operating systems.

65. (original) The method of claim 63, wherein operating system and the interrupted thread execute in different instruction set architectures of the computer.

66. (original) The method of claim 56, wherein the operating system and the thread execute in different execution modes of the computer, and the steps to maintain the association between the thread and the context are automatically invoked, without explicit software request, on a transition between the thread execution mode and the operating system execution mode.

67. (original) The method of claim 66, wherein the thread execution mode and the operating system execution mode are two different instruction set architectures of the computer.

68. (original) The method of claim 56, wherein the operating system maintains an association between contexts and corresponding threads of execution, each such context including values of data registers, the method further comprising:

modifying at least half of the data registers of the portion of the thread context maintained by the operating system before delivering the thread to the operating system.

69. (original) The method of claim 68, wherein at least some of the modified registers are overwritten by information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

70. (original) The method of claim 68, wherein at least some of the modified registers are overwritten by a value that enables validation of the contents of the context.

71. (original) The method of claim 56, further comprising the step of modifying a linkage return address for the thread to include information used to restore the context of the thread.

72. (original) The method of claim 71, wherein the linkage register is modified with information indicating an execution path by which, or a condition on which, execution arrived at the entry handler.

73. (original) The method of claim 71, wherein the modification leaves at least half of the bits of the linkage return address intact.

74. (original) The method of claim 71, wherein the linkage register is modified with information indicating a storage location at which at least the portion of the thread context to be modified is saved before the modifying.

75. (previously presented) The method of claim 56:

wherein the interrupted thread at the point of interruption executes in one instruction set architecture and the operating system is coded primarily in a different instruction set architecture; and

further comprising the step of setting of a register to a value that specifies actions to be taken by the entry handler or exit handler to convert operands from one form to another to conform to a data storage convention of the operating system instruction set architecture.

76. (original) The method of claim 56, further comprising either the step of deferring delivery of an interrupt before interrupting the thread by a time sufficient to allow the thread to reach a checkpoint in the execution of the thread where the amount of extended context, being the resources of the thread context beyond those whose resource association with the thread is maintained by the operating system, is reduced.

77. (original) The method of claim 56, further comprising either the step of rolling execution of the thread back to a checkpoint in the execution of the thread where the amount of extended context, being the resources of the thread context beyond those whose resource association with the thread is maintained by the operating system, is reduced.

78. (last amended 9/13/04) The method of claim 56, further comprising either the step of storing at least the extended context, the extended context being the resources beyond those whose resource association with the thread is maintained by the operating system, into a storage location, a pool of storage locations being managed by a queuing discipline in which empty storage locations in which a context is to be saved are allocated from the head of the queue, recently-emptied storage locations for reuse are enqueued at the head of the queue, and full storage locations to be saved are queued at the tail of the queue.

79. (original) A method, comprising:

during invocation of a service routine of a computer, passing a linkage return address to the service routine at which to resume execution on completion of the service, the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception;

on return from the service routine, attempting to execute the instruction at the linkage return address and raising the chosen exception; and

9 after servicing the exception, returning control to a caller of the service routine.

80. (original) The method of claim 79, wherein the passed linkage return address is selected to point to a memory page having a memory attribute that raises the chosen exception on an attempt to execute an instruction from the page.

81. (original) The method of claim 79, wherein

the service routine is an interrupt service routine of an operating system for a computer architecture other than the architecture native to the computer;

the service routine is invoked by an asynchronous interrupt; and

the caller is coded in the instruction set native to the architecture.

82. (previously presented) The method of claim 5, further comprising the step of:

without modifying a pre-existing thread scheduler of the computer, establishing an entry handler for execution at a specified entry point or on a specified entry condition to the thread scheduler, the entry handler programmed to save a context of an interrupted thread and modify the thread context before delivering the modified context to the thread scheduler.

83. (previously presented) The method of claim 20, further comprising the steps of:

during invocation of a service routine of the operating system, passing a linkage return address to the service routine at which to resume execution on completion of the service, the linkage return address being deliberately chosen so that an attempt to execute an instruction from the linkage return address on return from the service routine will raise a program execution exception;

on return from the service routine, attempting to execute the instruction at the linkage return address and raising the chosen exception; and

after servicing the exception, returning control to a caller of the service routine.